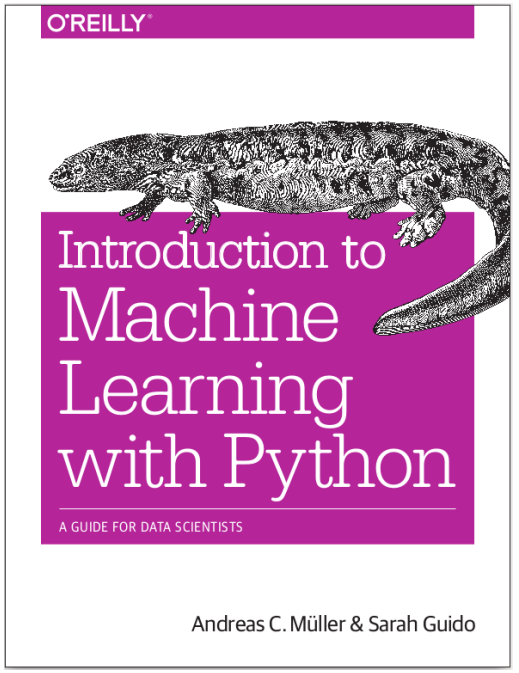
Building scikit-learn

Andreas Müller
Columbia University,
scikit-learn

universitätbonn ais

Amazon Machine Learning

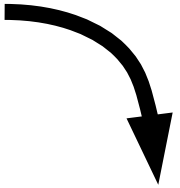


NYU Center for Data Science

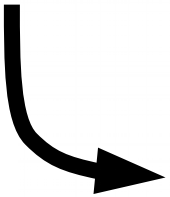
COLUMBIA UNIVERSITY
Data Science Institute

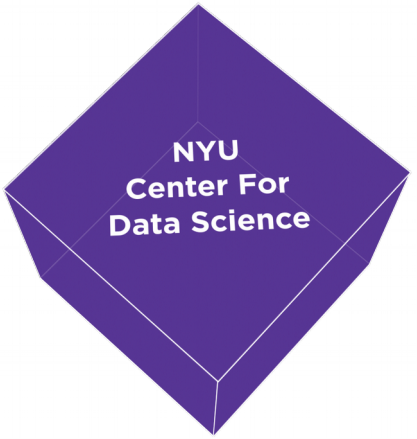
Me


universitätbonn
ais

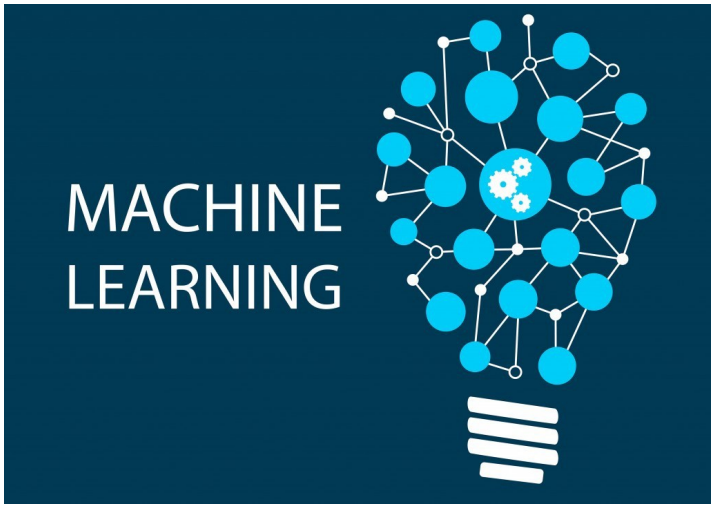



Amazon
Machine
Learning




NYU
Center For
Data Science


python


MACHINE
LEARNING

Classification
Regression
Clustering
Semi-Supervised Learning
Feature Selection
Feature Extraction
Manifold Learning
Dimensionality Reduction
Kernel Approximation
Hyperparameter Optimization
Evaluation Metrics
Out-of-core learning
……


Alexandre Gramfort
agramfort


Alexander Fabisch
AlexanderFabisch


Alexandre Passos
alextp


Andreas Mueller
amueller


Arnaud Joly
arjoly


Brian Holt
bdholt1

bthirion
bthirion


Chris Filo Gorgolewski
chrisfilo


David Cournapeau
cournape


Duchesnay
duchesnay


David Warde-Farley
dwf


Fabian Pedregosa
fabianp


Gael Varoquaux
GaelVaroquaux


Gilles Louppe
glouppe


Jake Vanderplas
jakevdp


Jaques Grobler
jaquesgrobler


Jan Hendrik Metzen
jmetzen


Jacob Schreiber
jmschrei


Joel Nothman
jnothman


Kyle Kastner
kastnerkyle


Lars
larsmans


Loïc Estève
lesteve


Shiqiao Du
lucidfrontier45


Mathieu Blondel
mblondel


Manoj Kumar
MechCoder


Noel Dawe
ndawe


Nelle Varoquaux
NelleV


Olivier Grisel
ogrisel


Paolo Losi
paolo-losi


Peter Prettenhofer
pprett


(Venkat) Raghav (Rajagopalan)
raghavrv


Robert Layton
robertlayton


Ron Weiss
ronw


Satrajit Ghosh
satra


sklearn-ci


sklearn-wheels


Tom Dupré la Tour
TomDLT


Vlad Niculae
vene


Virgile Fritsch
VirgileFritsch


Vincent Michel
vmichel

Wei Li
weilinear

Yaroslav Halchenko
yarikoptic

Mission

Commoditize and Democratize Machine Learning

Basic API

Representing Data

one sample

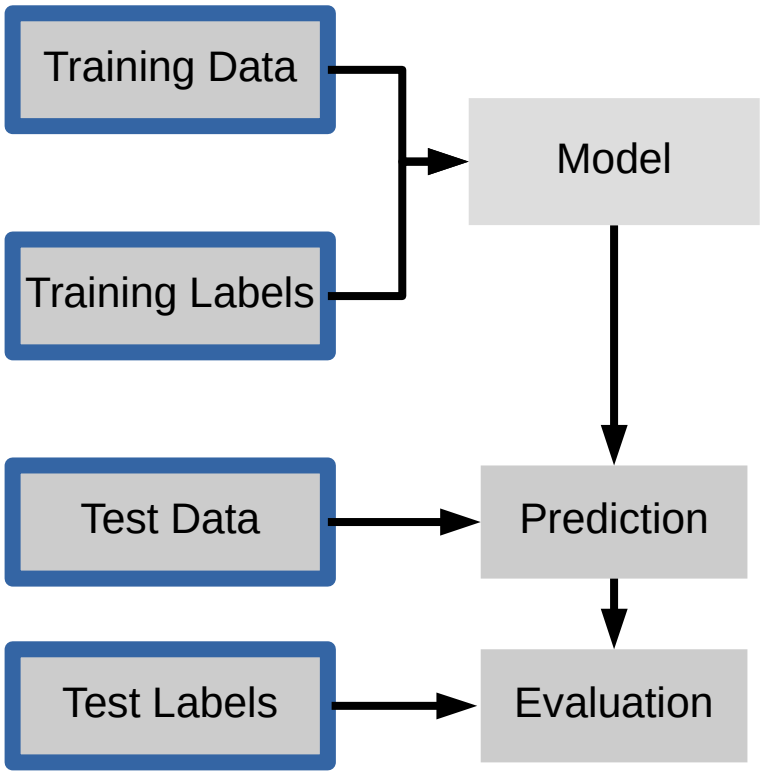
$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix}$$

one feature

$$y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

outputs / labels

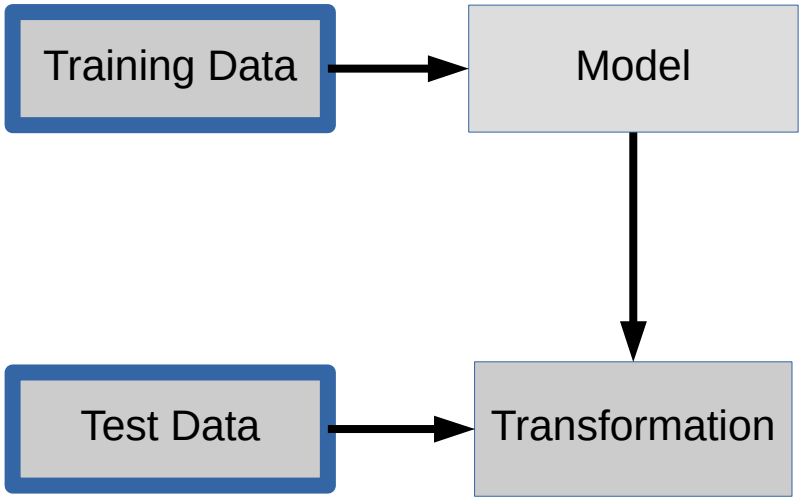
```
clf = RandomForestClassifier()

clf.fit(X_train, y_train)
```




```
y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```

Unsupervised Transformations

```
pca = PCA()

pca.fit(X_train)

X_new = pca.transform(X_test)
```



Core API Summary

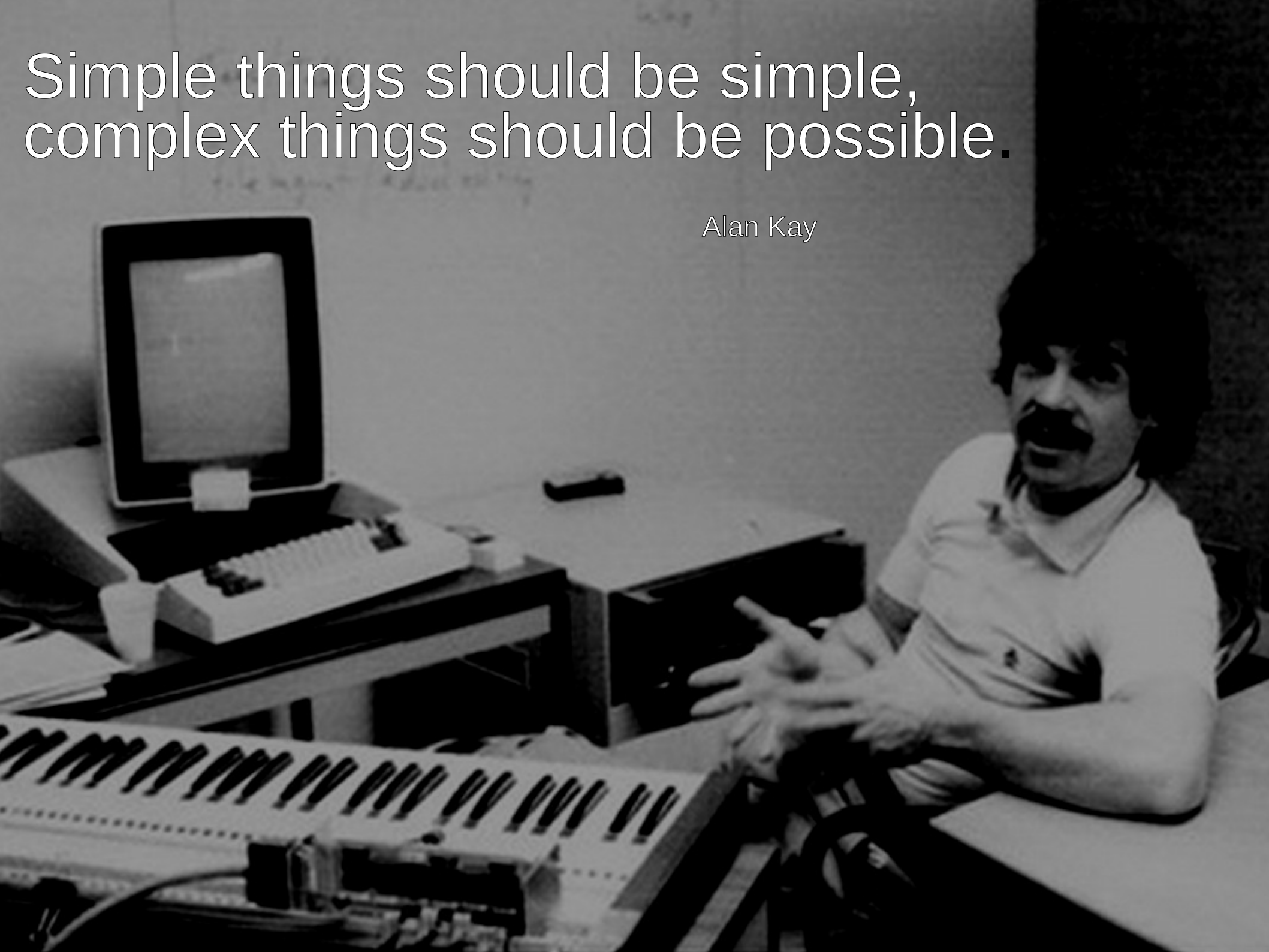
`estimator.fit(X, [y])`

`estimator.predict`	**`estimator.transform`**
Classification	Preprocessing
Regression	Dimensionality reduction
Clustering	Feature selection
	Feature extraction

Guiding Ideas

Goals:

Maintainability
Ease of use


Simple things should be simple,
complex things should be possible.
Alan Kay

Three way documentation

1.9. Ensemble methods

The goal of **ensemble methods** is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Two families of ensemble methods are usually distinguished:

- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

  Examples: *Bagging methods*, *Forests of randomized trees*, ...

- By contrast, in **boosting methods**, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

  Examples: *AdaBoost*, *Gradient Tree Boosting*, ...

`sklearn.ensemble`.RandomForestClassifier

class `sklearn.ensemble.` **`RandomForestClassifier`** (*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

Parameters:

n_estimators : integer, optional (default=10)

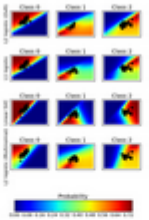
The number of trees in the forest.

criterion : string, optional (default="gini")

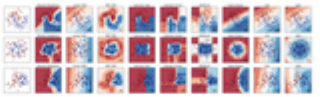
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

Examples

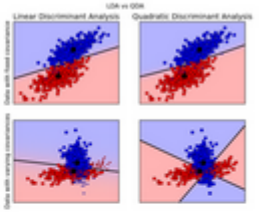
Recognizing hand-written digits



Plot classification probability



Classifier comparison



Linear and Quadratic Discriminant Analysis with confidence ellipsoid

Ease of Installation

Simplicity

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```

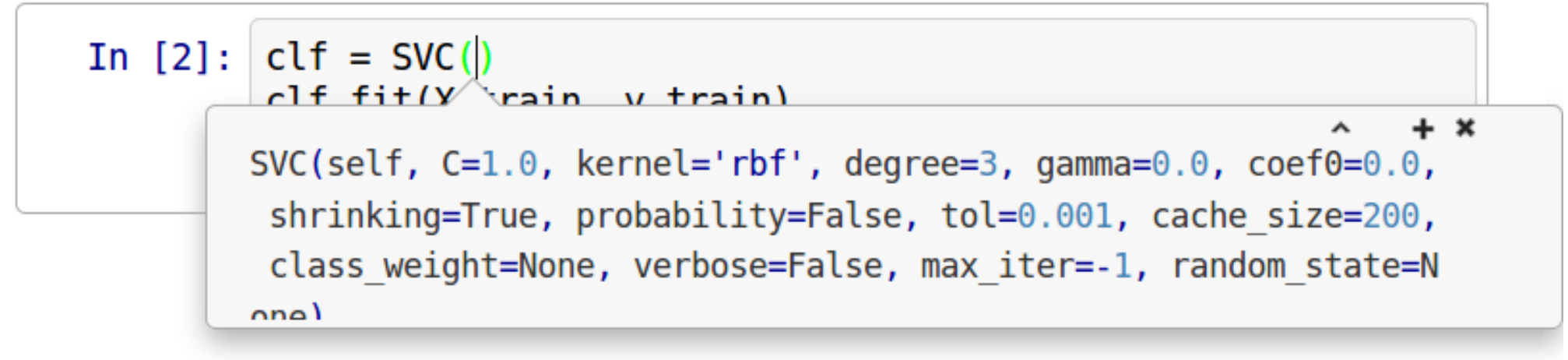
Consistency

```
grid = GridSearchCV(svm,param_grid)
grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

Composition

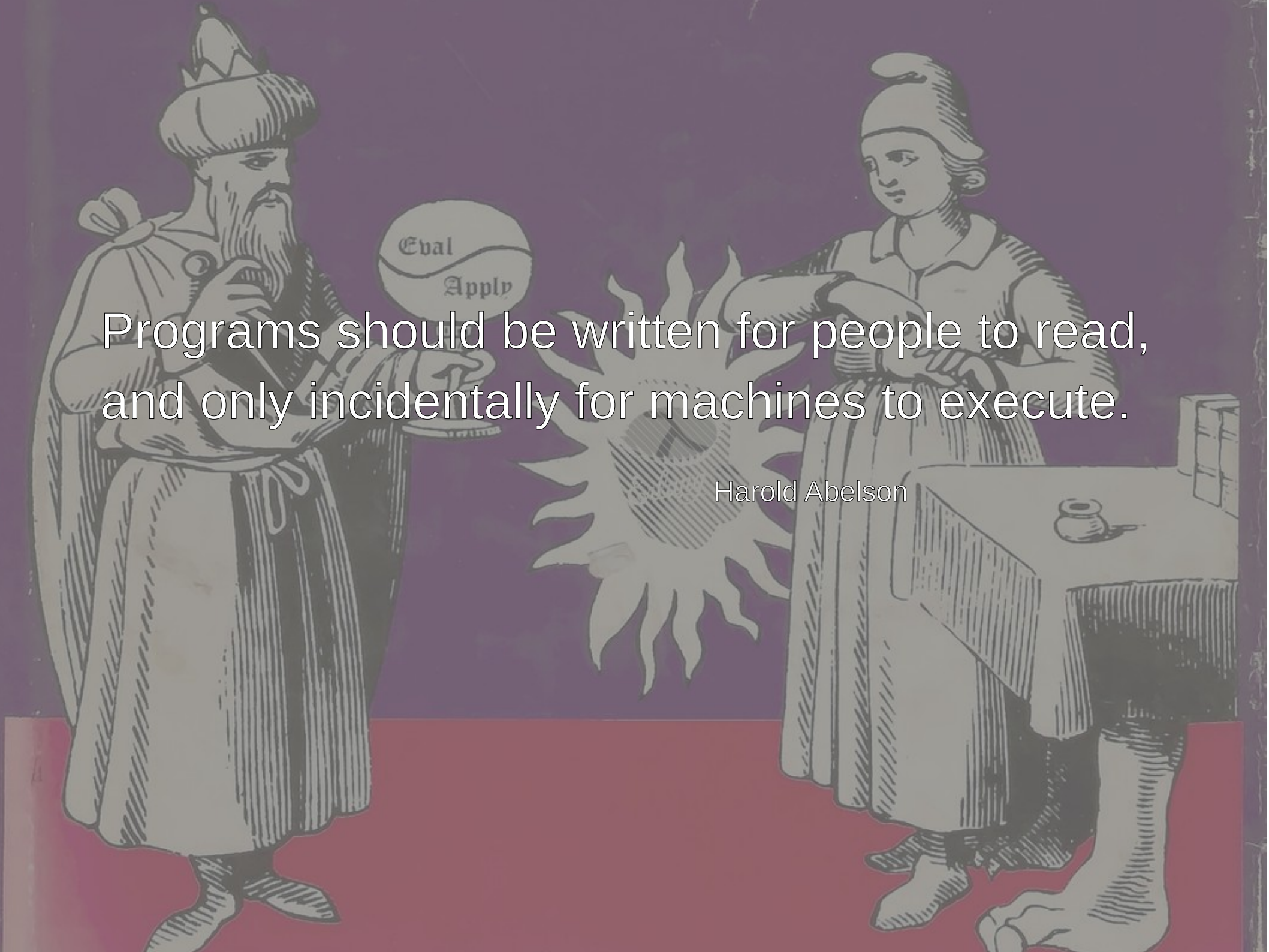
```
feature_selection = RFECV(NaiveBayes())
pipe = make_pipeline(feature_selection,
                     RandomForestClassifier())
grid = GridSearchCV(pipe, param_grid)
```

Default Parameters



Flat Class Hierarchy, Few Types

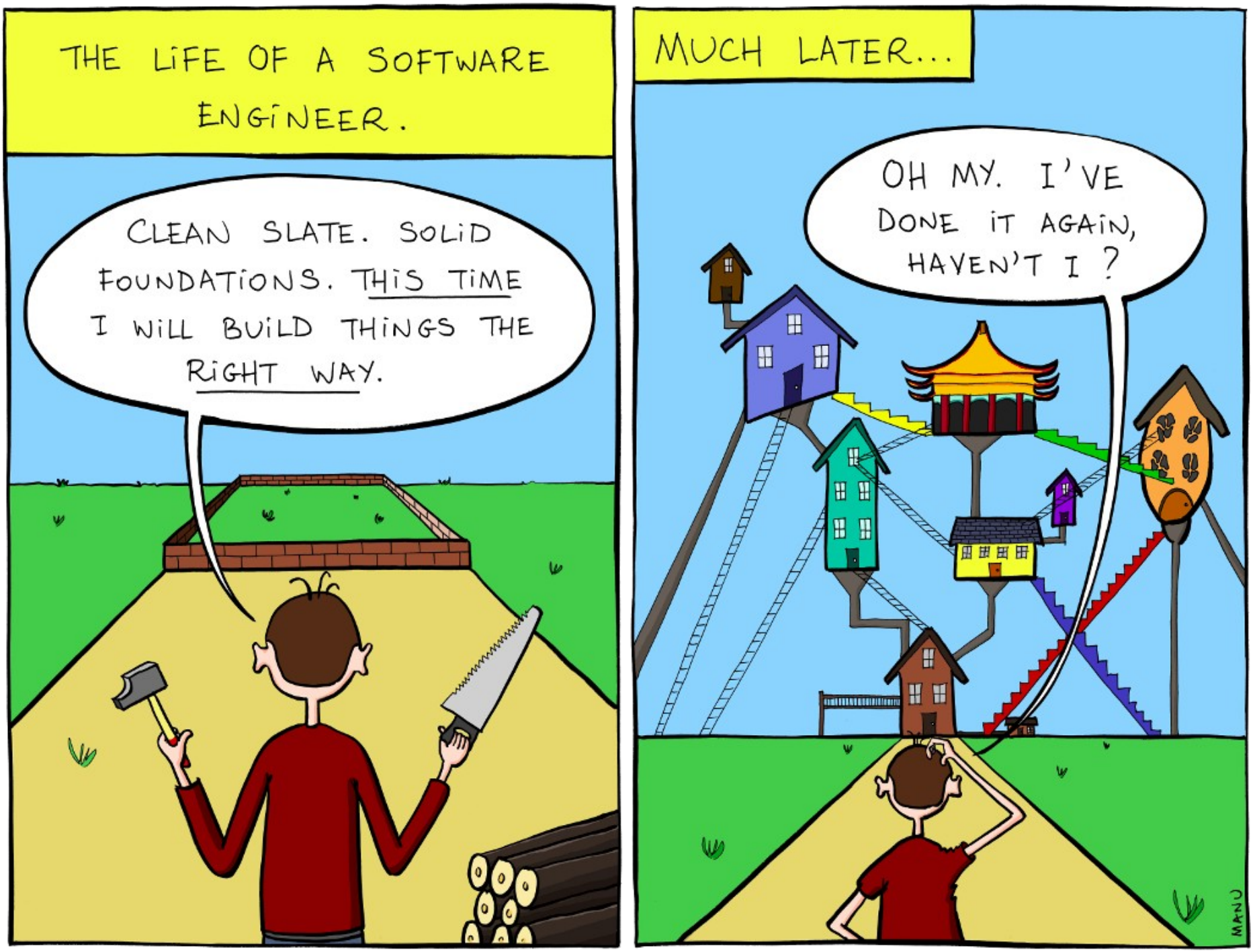
- Numpy arrays / sparse matrices
- Estimators
- [Cross-validation objects]
- [Scorers]



Programs should be written for people to read, and only incidentally for machines to execute.

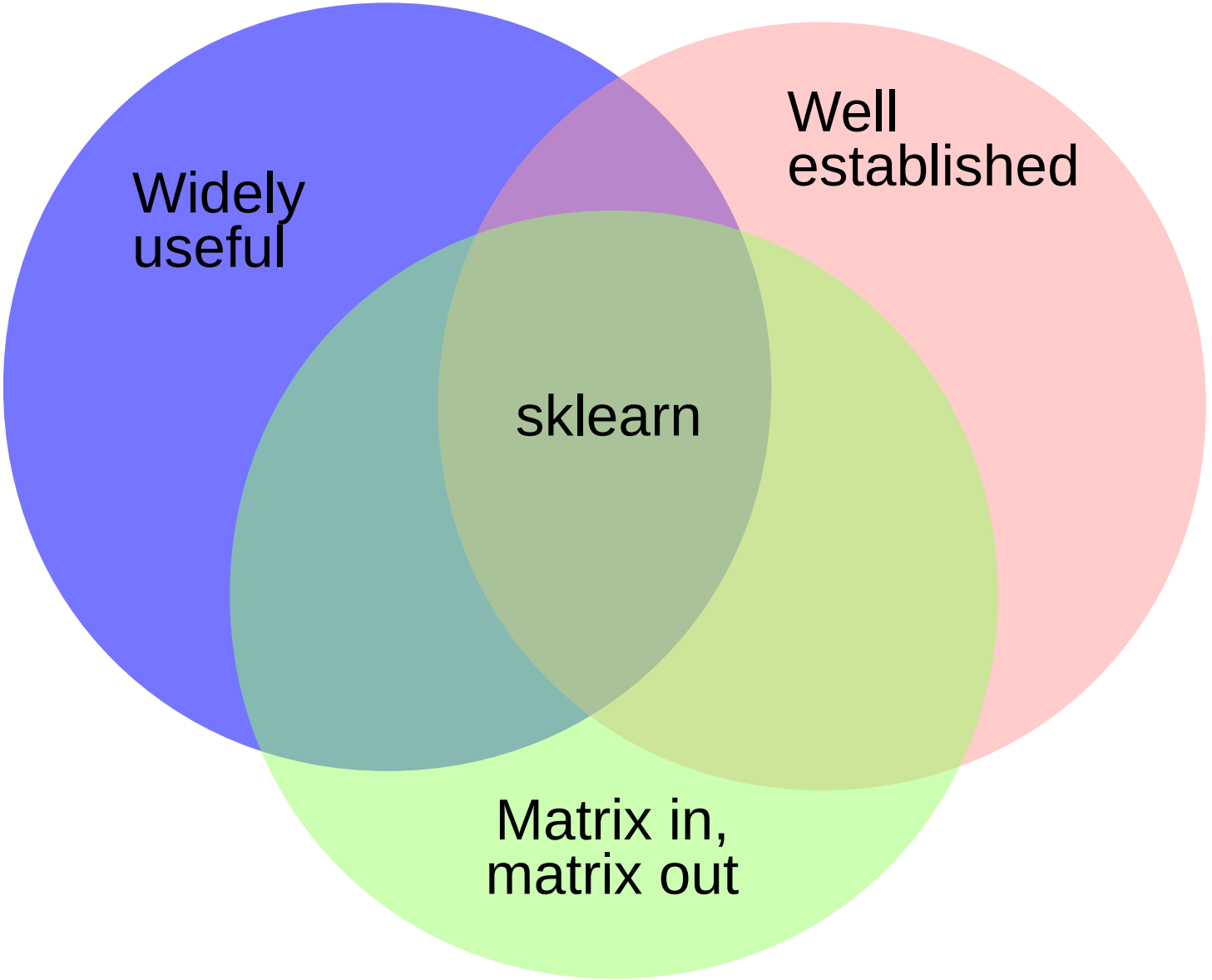
Harold Abelson

Maintainability

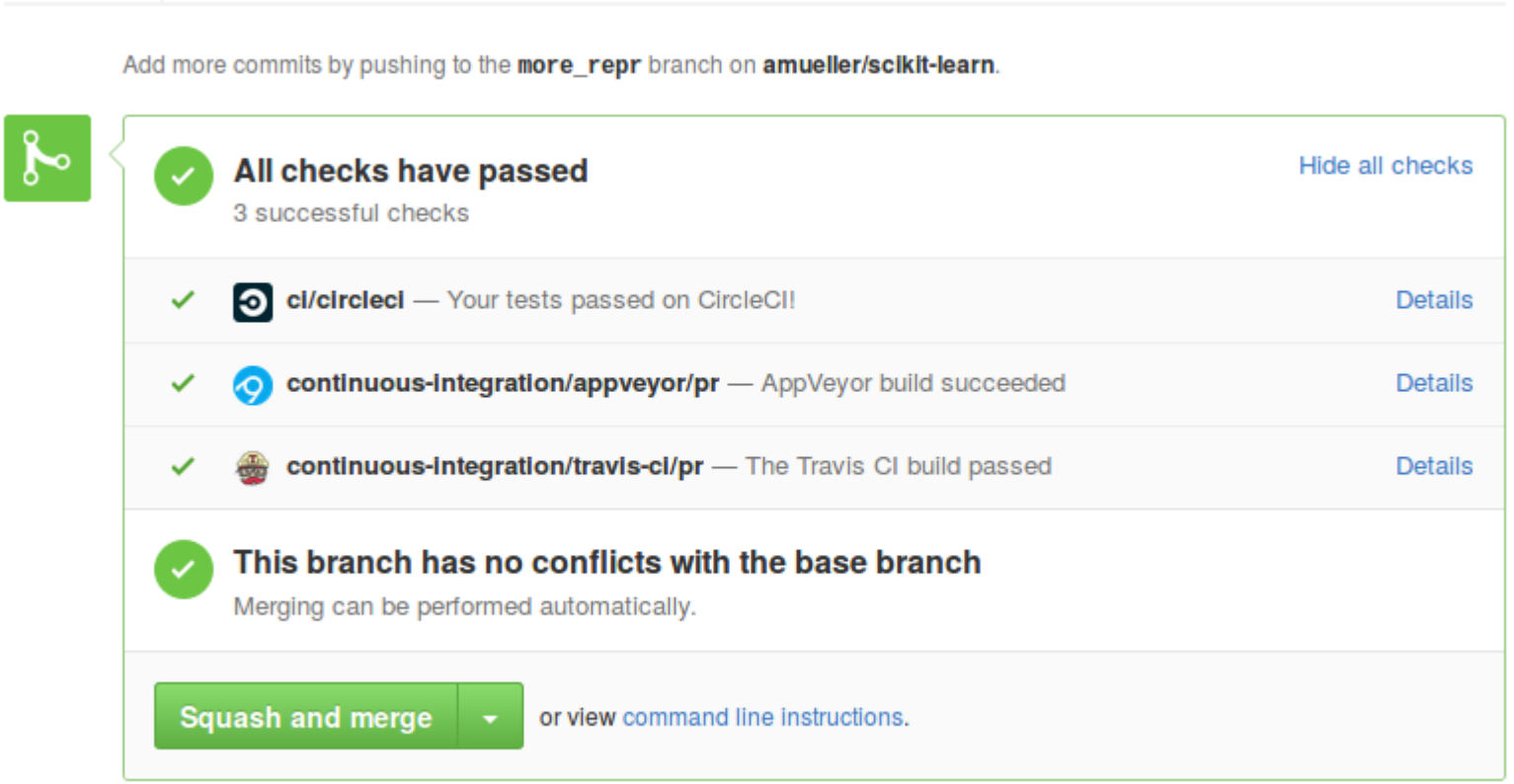


Feature Creep

Scoping


Widely useful
Well established
sklearn
Matrix in, matrix out

Testing & Continuous Integration



avoid code;
avoid code rot!

API implementation & contracts

Method Chaining

Fit must return self:

```
X_scaled = StandardScaler().fit(X).transform(X)

pred = SVC().fit(X_train, y_train).score(
    X_test, y_test)
```

fit resets

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=10)
tree.fit(iris.data, iris.target)
score_iris = tree.score(iris.data, iris.target)
tree.fit(digits.data, digits.target)
score_digits = tree.score(digits.data, digits.target)
```

(only) fit mutates self

- __init__ only remembers construction parameters
- Transform / score etc don’t change object
- Fit returns self but mutates object!

Estimated attributes

```
Attributes
----------
components_ : array, shape (n_components, n_features)
    Principal axes in feature space, representing the directions of
    maximum variance in the data. The components are sorted by
    ``explained_variance_``.

explained_variance_ : array, shape (n_components,)
    The amount of variance explained by each of the selected components.

    Equal to n_components largest eigenvalues
    of the covariance matrix of X.

    .. versionadded:: 0.18

explained_variance_ratio_ : array, shape (n_components,)
    Percentage of variance explained by each of the selected components.

    If ``n_components`` is not set then all components are stored and the
    sum of the ratios is equal to 1.0.

singular_values_ : array, shape (n_components,)
    The singular values corresponding to each of the selected components.
    The singular values are equal to the 2-norms of the ``n_components``
    variables in the lower-dimensional space.
```

Leaves two kinds of attributes:
arguments to __init__; things estimated during fit

fit_transform / fit_predict

In general: computational shortcut:

```
pca = PCA()
pca.fit(X)
X_pca = pca.transform(X)
X_pca2 = pca.fit_transform(X)
```

Clustering / Manifold learning: Not inductive.

```
tsne = TSNE()
X_tsne = tsne.fit_transform(X)
```

```
dbscan = DBSCAN()
cluster_labels = dbscan.fit_predict(X)
```

check_estimator

```python
class TemplateClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self, demo_param='demo'):
        self.demo_param = demo_param

    def fit(self, X, y):

        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        # Return the classifier
        return self

    def predict(self, X):

        closest = np.argmin(euclidean_distances(X, self.X_), axis=1)
        return self.y_[closest]
```

```
check_estimator(TemplateClassifier)
```

```
AssertionError: Error message does not include the expected string: 'fit'.
Observed error message: "'TemplateClassifier' object has no attribute 'X_'"
```

Development Practices

Standards for OSS

Everything discussed in the open.
Every convention and process documented.

Development guide

http://scikit-learn.org/dev/developers/contributing.html

Contains:

- API details
- Bug report guidelines
- PR guidelines
- Reviewing guidelines
- How to find issues
- Details of CI

Even more at http://scikit-learn.org/dev/developers/

Deprecations / backward compatibility

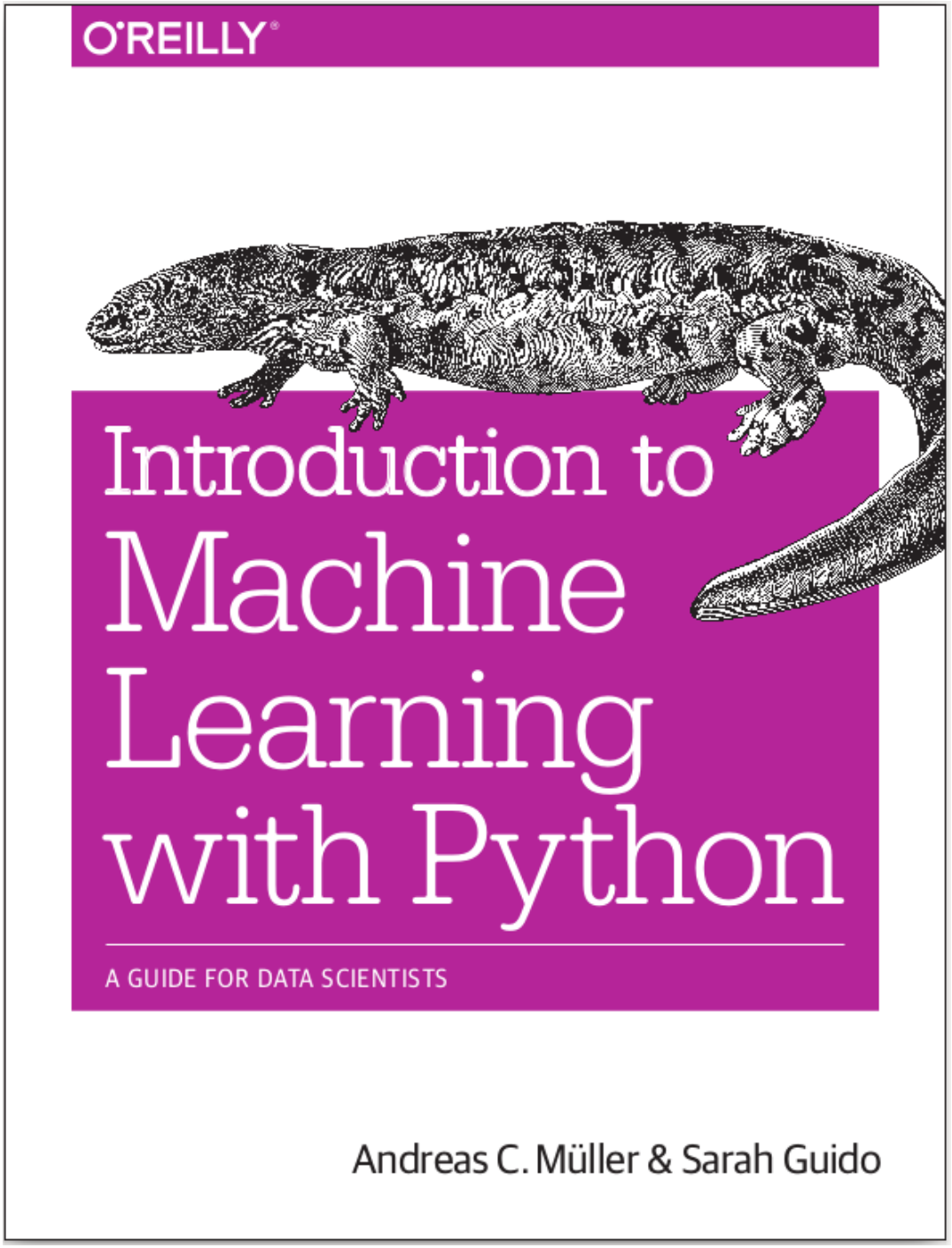
- Don’t change any behavior (except bug fixes)

```
@property
@deprecated("Attribute labels_ was deprecated in version 0.13 and "
            "will be removed in 0.15. Use 'classes_' instead")
def labels_(self):
    return self.classes_
```

(my personal) Roadmap

Make simple things simple again!

Missing Values
Categorical Variables
Plotting Tools
Pandas Integration
Feature Names



amueller.github.io

@amuellerml

@amueller

t3kcit@gmail.com